

Technical Article

Intro to Game

Lab version: 1.0.0

Last updated: 10/6/2011

CONTENTS

OVERVIEW	3
THE XNA GAME LIFE CYCLE	4
CONTENT MANAGEMENT	8
XNA GAME STUDIO	9
SUMMARY	11

Overview

Designing the architecture for a game and programming it may be an intimidating task for the novice programmer.

The XNA Framework exists solely for making game development an easier task, creating a working skeleton of a game for the programmer, and allowing the programmer to focus on the game logic and content resources. XNA games are able to run under Windows Phone7, Xbox 360, and PCs running a Windows-based operating system.¹

In this article the XNA game life cycle is introduced, content and resource management is discussed, and how develop a game development by using XNA Game Studio 4.0 is presented.

Objectives

When you finish reading the article, you will have:

- A high-level understanding of the XNA Game Loop and the XNA Game Studio.
- A familiarity with the content and content manager ideas.

¹ Only Windows XP with SP3 (and newer), Vista with SP2 (and newer), and Windows 7 (all versions) are supported.

The XNA Game Life Cycle

Game Steps and Stages

A running game usually goes through the following steps:

1. **Initialization/Load** – Sets default and preliminary values to the game, queries and initializes user-based information, loads graphic and non-graphic contents, et cetera.
2. **The Game Loop** – Performs in-game repeating logic and layout calculations and render.
3. **Unload/Shutdown** – Saves current state, releases and unloads contents, et cetera.

XNA provides a skeleton that is valid for any type of game. The skeleton is implemented within the **Game** class, which is a part of the Microsoft.Xna.Framework namespace. Once inherited, the **Game** class provides the complete game life cycle, enabling the programmer to override the required methods in order to add specific game logic.

Game Class Methods

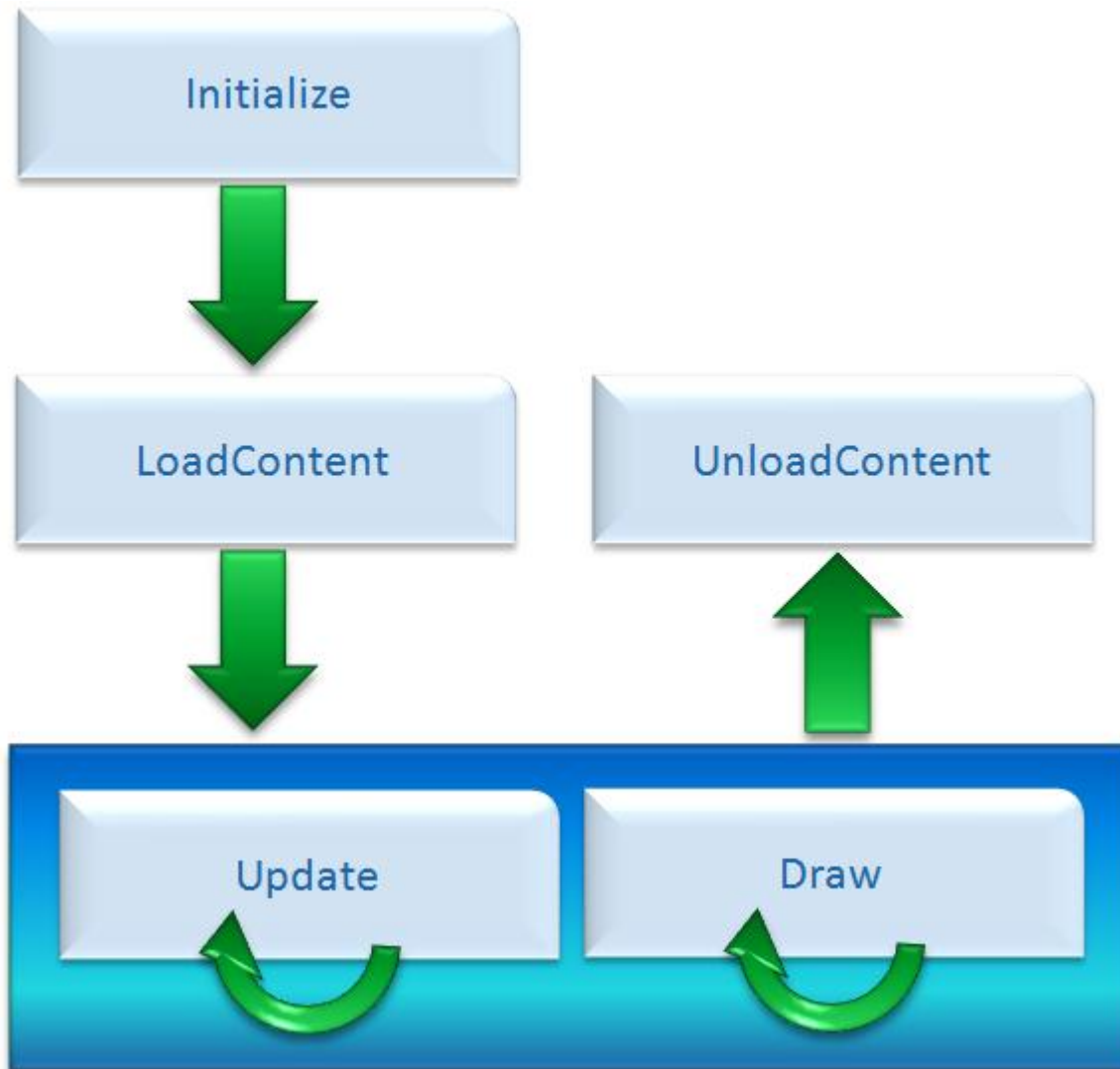
The **Game** class contains, among other things, virtual methods that cover the various aspects of the listed game steps. An inherited **Game** class may override some or all of these methods, and define a constructor as described below.

- **Class constructor** – Used to instantiate and set default values to required elements. For example, instantiate the graphics device manager, define the game frame rate, et cetera.
- **Initialize** – Sets default and preliminary values to the game shell, queries and initializes user-based information, et cetera.
- **LoadContent** – Loads all graphics and other content required to run the game. For example, **LoadContent** loads and instantiates graphic sprite batches, background images, sounds, and so on.
- **Update** – Performs on-going game logic: calculates current positions, physics, collisions and states; collects input information from the various input devices; plays audio, and so on.
- **Draw** – Draws the current view of the game: backgrounds, sprites, et cetera.
- **UnloadContent** – Unloads all game content and content managers.

Game Life Cycle

The XNA Framework is in charge of running the programmed game. The framework calls, when appropriate, the inherited **Game** class object methods as shown in Figure1.

Figure 1. The game loop, as run automatically by the XNA framework for a Game class inherited object



The **Initialize** method is called at game startup to allow the game to do any initialization required by the game shell itself. **Initialize** is followed by a single call to the **LoadContent** method, which allows the game to load all required content resources as described above.

The **Update** and **Draw** methods are called repeatedly by the XNA Framework—not necessarily in sequence, but multiple times every second—according to the frame rate defined in the initialization phase. For more information, see [Frame Rates](#).

Because the **Update** method override is responsible for the on-going game logic, perform the following actions when overriding it:

- Gather input information: gestures, multitouch, et cetera.
- Update physics and AI for the various game elements according to the current game state and the gathered input information.
- Update animations. Note that at this stage you only decide upon the current frame to display. Drawing is not performed because all drawings should be done by the **Draw** override method.
- Update camera. Refer to the update animations note, which also applies to updating the camera. A change caused by a camera update is not rendered.

As stated, the **Draw** method override should draw all visible screen elements. The **UnloadContent** method is called once when the game closes to allow the game to release loaded resources.

The typical pattern of game development should be that all input, game logic (including on-screen elements' physics), and AI or any other nongraphical processing should be applied from within the **Update** method. All graphical processing and actual drawing of the game should be done by the **Draw** method override.

Frame Rates

Frame rates for PC games are limited by the computer's graphics card and CPU hardware capabilities. Conversely, Xbox 360 games operate at a rate of up to 60 frames per second (fps). By default, XNA Framework for Windows Phone 7 titles attempt to render at 30 fps.

The XNA Framework is designed to drop frames automatically in order to keep up with the desired frame rate. There may be cases where **Draw** is not called even though **Update** changed the elements to be rendered, expecting the elements to be drawn again on the next cycle of **Draw**.

This process occurs because updating the course of the game is more important than drawing it. A few frames may be drawn because of a lack of computation resources; nevertheless, it is important to understand where you are in the game. If **Update** calls are dropped, game logic may not only be affected, but also may be damaged.

Drawing and Graphics

In order to draw the game, the programmer requires a graphics device that acts as the link between the program and the device's screen.

The XNA Framework provides the **GraphicDeviceManager** class, which is a part of the `Microsoft.Xna.Framework` namespace. When the namespace is instantiated with an instance to a **Game** class inherited object as a parameter, it creates a **GraphicsDevice** that allows the **Game** class instance to draw the game screen to the device.

For Windows Phone 7 games, the **GraphicDeviceManager** (GDM) instance already defaults to a size of 800x480 pixels as the drawing surface available to the programmer, and uses a **GraphicsDevice** class instance to do the drawing.

To draw sprite elements, the programmer has to instantiate the **SpriteBatch** class (part of the Microsoft.Xna.Framework.Graphics namespace) and to use it from within the game's **Draw** method to draw textures and sprites. Drawing operations performed with a **SpriteBatch** instance have to be preceded by a call to the **Begin** method, and followed by a call to the **End** method. Several operations should occur between calls to **Begin** and **End**. These methods are used to save on draw calls made by the **SpriteBatch** instance to **GraphicsDevice** instance.

All of this is performed to prepare the buffers for on-screen display. Preparation is accomplished by the **Game** class instance calling the **Present** method on the enclosed **GraphicsDevice** instance after the **Draw** method, meaning after all elements finish their rendering processes.

Textures and sprites are available using objects of the **Texture2D** class (a part of the Microsoft.Xna.Framework.Graphics namespace) that can be loaded from the game content manager, as described further in this article.

Content Management

Games, much like any other application, are made of content and logic. While logic is handled from within the game class **Update** override method, the game content has to be managed separately.

Using the **ContentManager** Class

The XNA framework provides the **ContentManager** class, which is a part of the `Microsoft.Xna.Framework.Content` namespace. An instance of the **ContentManager** class allows the programmer to define a root directory for the content resources, and to load resources into the game.

Content resources are files—such as graphic files and sound files—that are essential to game execution.

Managing Content Batches

The **Game** class holds a reference to a **ContentManager** within the `Content` property. Redefining this property to different instances of **ContentManager** allows the programmer to load different sets of content resources easily.

The typical pattern implemented by using the **ContentManager** class occurs when loading a different resource batch to a **ContentManager** for each different game level, and re-referencing the `Game.Content` property accordingly.

XNA Game Studio

While the XNA Framework encapsulates the game, the programmer has to repeat the process of creating the inheriting game class, defining and overriding the required methods, and ensuring that nothing is forgotten along the way.

Visual Studio 2010 as a Game Studio

Visual Studio 2010 acts as a game studio. It provides a set of templates that enables creation of various types of games for such Windows Phone 7, Windows, Xbox 360, et cetera.

Windows Phone 7 Game Template

Upon selecting and creating a Windows Phone 7 game by using the Windows Phone Game (4.0) template, a complete solution is generated. The generated solution holds two projects: the game project holding the game code and the content project holding the content resources of the game. Inside the game project, a class derived from the **Game** class is generated, already implementing the following:

1. **GraphicsDeviceManager** and **SpriteBatch** fields.
2. Constructor – Instantiating the **GraphicsDeviceManager** field, defining a default root directory for the game content, and setting the frame rate to 30 fps for a Windows Phone 7 game.
3. **Initialize** override – Containing a placeholder for game initialization and calling the `base.Initialize` method.
4. **LoadContent** override – Instantiating the **SpriteBatch** field and leaving a placeholder for game content loading.
5. **Update** override – Testing for the **Back** button being pressed and exiting the game accordingly, leaving a placeholder for the in-game logic update.
6. **Draw** override – Clearing the surface with a blue background, and leaving a placeholder for further drawing instructions.
7. **UnloadContent** override – A placeholder for game content unloading.

Complete Game Loop Generated

Creating an XNA game by using the Visual Studio 2010 template generates the complete game loop, and creates a running game with a single click. The programmer's task is to fill in the correct spots to create the desired game.

Summary

Programming an XNA game for Windows Phone 7 can be an easy task when using XNA Game Studio 4.0. The XNA Framework takes care of many of the common tasks that need to be handled by the programmer developing a game. It allows the programmer to concentrate on the game itself. Visual Studio 2010 acts as a game studio, providing the template of a complete game skeleton and the environment to program it.

See the Windows Phone 7 Game State Management sample at <http://creators.xna.com/en-US/sample/phonegamestatemanagement> to find out how to define game screens, manage game states, et cetera.